Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

Fast Gradient Descent for Drifting Least Squares Regression, with Application to Bandits

Nathan Korda
MLRG, University of Oxford, UK.

Prashanth L.A. and **Rémi Munos**
INRIA Lille - Nord Europe, Team SequeL, FRANCE.

Abstract

Online learning algorithms require to often recompute least squares regression estimates of parameters. We study improving the computational complexity of such algorithms by using stochastic gradient descent (SGD) type schemes in place of classic regression solvers. We show that SGD schemes efficiently track the true solutions of the regression problems, even in the presence of a drift. This finding coupled with an $O(d)$ improvement in complexity, where d is the dimension of the data, make them attractive for implementation in the *big data* settings. In the case when strong convexity in the regression problem is guaranteed, we provide bounds on the error both in expectation and high probability (the latter is often needed to provide theoretical guarantees for higher level algorithms), despite the drifting least squares solution. As an example of this case we prove that the regret performance of an SGD version of the PEGE linear bandit algorithm is worse than that of PEGE itself only by a factor of $O(\log^4 n)$. When strong convexity of the regression problem cannot be guaranteed, we investigate using an adaptive regularisation. We make an empirical study of an adaptively regularised, SGD version of LinUCB in a news article recommendation application, which uses the large scale news recommendation dataset from Yahoo! front page. These experiments show a large gain in computational complexity and a consistently low tracking error.

Introduction

Often in learning algorithms an unknown parameter must be estimated from data arriving sequentially in pairs, (x_n, y_n). We consider settings where the points x_n are chosen by a higher level algorithm and the outputs y_n satisfy the dynamics $y_n = x_n^\intercal \theta^* + \xi_n$, where ξ_n is i.i.d., zero-mean noise, and θ^* is the unknown parameter (the flow diagram, Fig. 1, illustrates this setting). Typically, in such cases an ordinary least squares (OLS) estimate is used for θ^*, and finding this estimate is often the most computationally intensive part of the higher level algorithm. The solution to the least squares regression problem is defined as

$$\hat{\theta}_n = \arg\min_{\theta} \left\{ F_n(\theta) := \frac{1}{2} \sum_{i=1}^{n} (y_i - \theta^\intercal x_i)^2 \right\}. \qquad (1)$$

That $\hat{\theta}_n = \bar{A}_n^{-1} \bar{b}_n$, where $\bar{A}_n = n^{-1} \sum_{i=1}^{n} x_i x_i^\intercal$ and $\bar{b}_n = n^{-1} \sum_{i=1}^{n} x_i y_i$, is well-known. Assuming that the features x_i evolve in a compact subset $\mathcal{D}$ of $\mathbb{R}^d$, the complexity of solving (1) with the above approach is $O(d^2)$, where the inverse of $\bar{A}_n$ is computed iteratively using the Sherman-Morrison lemma. Using the Strassen algorithm or the Coppersmith-Winograd algorithm gives a complexity of $O(d^{2.807})$ and $O(d^{2.375})$ respectively. In addition, there is an order $O(d^2 n)$ complexity for computing $\bar{A}_n$.

Unlike the traditional gradient descent (GD) setting where the pairs (x_n, y_n) are samples drawn from some unknown joint probability distribution, we assume that the samples, x_n, are chosen by a higher level learning algorithm, and the problem is to find a good enough approximation to θ^* for its purposes, given these non-i.i.d. samples. This poses a new difficulty in applying GD schemes directly, and we outline two well-known solutions to this problem in the following.

As illustrated in Fig. 1, the classic SGD algorithm operates by maintaining an iterate θ_n that is updated as follows: Choose a random sample (x_{i_n}, y_{i_n}), where i_n is picked uniformly at random in $\{1, \ldots, n\}$ and update

$$\theta_n = \theta_{n-1} + \gamma_n (y_{i_n} - \theta_{n-1}^\intercal x_{i_n}) x_{i_n}, \qquad (2)$$

(The sequence of stepsizes γ_n is chosen in advance, see assumption (A4) below for details.) The complexity of each iteration above is $O(d)$, while traditional approaches giving the exact solution, such as using the Sherman-Morrison lemma, incur a cost of at least $O(d^2)$ per iteration. We shall refer to SGD applied to our setting as fOLS-GD (fast Online Least Squares - Gradient Descent).

Unlike previous works which analyse the above SGD algorithm in a batch setting, we consider a drifting least squares setting. In particular, at each instant n, the SGD update is required to track the minimiser $\hat{\theta}_n$ of the function $F_n(\cdot)$, as n increases. The practical advantage of such an approach is to replace the costly inversion of the $\bar{A}_n$ matrix with an efficient iterative scheme. However, from a theoretical standpoint, fOLS-GD has to grapple with the drift error, $\|\hat{\theta}_n - \hat{\theta}_{n-1}\|_2$, that accumulates with time.

Under a minimum eigenvalue assumption on the matrices $\bar{A}_n$, we find that ordinary SGD is sufficient to mitigate the effects of drift in $\hat{\theta}_n$. In this case, we provide bounds both in expectation and in high probability on the approximation error $\theta_n - \hat{\theta}_n$, where θ_n is the fOLS-GD iterate at instant n (see

Theorem 1). Such bounds are essential for giving theoretical guarantees when using fOLS-GD as a subroutine to replace the matrix inversion approach to the regression problem in a higher level learning algorithm.

To cope with situations where the minimum eigenvalue assumption of the $\bar{A}_n$ matrix cannot be guaranteed by the higher level algorithm we propose adding an adaptive regularisation: since our data is growing with time we introduce a regularisation parameter, λ_n, that adapts to the sample size n as follows:

$$\tilde{\theta}_n := \arg\min_{\theta} \frac{1}{2n} \sum_{i=1}^{n} (y_i - \theta^\intercal x_i)^2 + \lambda_n \|\theta\|_2^2 . \quad (3)$$

This algorithm, which we henceforth refer to as fRLS-GD (fast Regularised online Least Squares - Gradient Descent), tracks the regression solutions, $\tilde{\theta}_n$ and operates in a manner similar to fOLS-GD (see Fig. 1) except that we factor in the regularisation parameter λ_n into the update rule:

$$\theta_n = \theta_{n-1} + \gamma_n((y_{i_n} - \theta_{n-1}^\intercal x_{i_n})x_{i_n} - \lambda_n \theta_{n-1}). \quad (4)$$

Unlike fOLS-GD, the above algorithm will suffer a bias due to the adaptive regularisation and it is difficult to provide bounds in theory owing to the bias error (see discussion after Eq. (10)). However, we demonstrate empirically that fRLS-GD is able to consistently track the true RLS solutions, when used within a higher level algorithm. The advantage, however, of using fRLS-GD in place of classic RLS solvers is that it results in significant computational gains.

As examples of higher level learning algorithms using regression as a subroutine, we consider two linear bandit algorithms. In a linear bandit problem the values x_n represent actions taken by an agent and the values $y_n = x_n^\intercal \theta^* + \xi_n$ are interpreted as random rewards, with unknown parameter θ^*. At each time the agent can choose to take any action $x \in \mathcal{D}$, where $\mathcal{D}$ is some compact subset of $\mathbb{R}^d$, and the agent's goal is to maximise the expected sum of rewards. This goal would be achieved by choosing $x_n = x^* := \arg\min_x \{x^\intercal \theta^*\}$, $\forall n$. However, since one does not know θ^* one needs to estimate it, and a tradeoff appears between sampling pairs (x_n, y_n) that will improve the estimate, and gaining the best short term rewards possible by exploiting the current information available. Typically the performance of a bandit algorithm is measured by its *expected cumulative regret*: $\mathcal{R}_n = \sum_{i=1}^{n} (x^* - x_i)^\intercal \theta^*$.

First, we consider the PEGE algorithm for linear bandits proposed by (Rusmevichientong and Tsitsiklis 2010). This algorithm is designed for action sets $\mathcal{D}$ satisfying a strong convexity property (see assumption (A4)), and so we can provide a computationally efficient variant of PEGE where the fOLS-GD iterate, θ_n, is used in place of the OLS estimate, $\hat{\theta}_n$, in each iteration n of PEGE. PEGE splits time into exploration and exploitation phases. During the exploitation phases the algorithm acts greedily using OLS estimates of θ^* calculated from data gathered during the exploration phases. During the exploration phases data is gathered in such a way that the smallest eigenvalues of $\bar{A}_n$ matrices are uniformly bounded for all n. The regret performance of this algorithm is $O(dn^{1/2})$, and we establish that our variant using fOLS-GD as a subroutine achieves an improvement of order $O(d)$ in complexity, while suffering a loss of only $O(\log^4 n)$ in the regret performance.

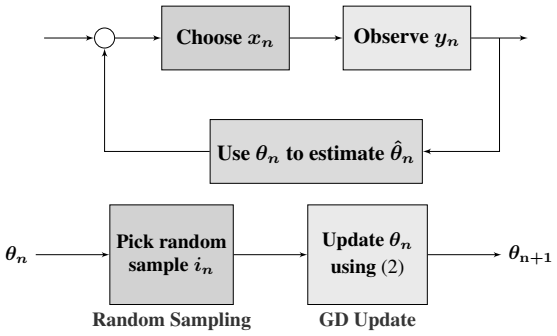


Figure 1: Estimating OLS $\hat{\theta}_n$ using online SGD within a higher-level machine learning algorithm

Second, we consider the LinUCB algorithm proposed (Li et al. 2010). Here we investigate computationally efficient variants of LinUCB. We begin by replacing the OLS estimate with an fRLS-GD iterate, and then compare this to two other state-of-the-art OLS schemes from (Johnson and Zhang 2013) and (Roux, Schmidt, and Bach 2012). The LinUCB algorithm is designed for situations where at each time, n, the agent can choose only from a given, finite subset of $\mathcal{D}$. The algorithm then calculates an optimistic upper confidence bound (UCB) for the mean reward associated with each feature, and then selects a feature greedily with respect to this UCB[1].

LinUCB, however, cannot guarantee that the minimum eigenvalue of $\bar{A}_n$ matrices is uniformly bounded, and so we apply fRLS-GD in place of fOLS-GD. Moreover, we devise a simple GD procedure for estimating the confidence term of the UCB for each arm. The resulting LinUCB variant achieves an $O(d)$ improvement in complexity over regular LinUCB. From the numerical experiments, we observe that the fRLS-GD iterate as well as SVRG (Johnson and Zhang 2013) and SAG (Roux, Schmidt, and Bach 2012) variants consistently track the true RLS solutions in each iteration of LinUCB, while the runtime gains are significant.

Related work. SGD is a popular approach for optimizing a function given noisy observations, while incurring low computational complexity. Non-asymptotic bounds in expectation for SGD schemes have been provided by (Bach and Moulines 2011). In the machine learning community, several algorithms have been proposed for minimising the regret, for instance, (Zinkevich 2003; Hazan and Kale 2011; Rakhlin, Shamir, and Sridharan 2011) and these can be converted to find the minimiser of a (usually convex) function. A closely related field is stochastic approximation (SA), and concentration bounds for SA algorithms have been provided by (Frikha and Menozzi 2012). Adaptive regulari-

[1]Calculating the UCBs is in itself an NP-hard problem for all but simple decision sets. However, we alleviate this problem by considering a setting where the sets of arms at each time instant is a finite subset of $\mathcal{D}$.

sation in the context of least squares regression has been analysed in (Tarrès and Yao 2011). For recent algorithmic improvements to solving batch problems, the reader is referred to the works of (Roux, Schmidt, and Bach 2012; Shalev-Shwartz and Zhang 2012; Johnson and Zhang 2013).

In general, none of the schemes proposed above are directly applicable in our setting due to two difficulties:
(i) our data $\{(x_i, y_i)\}_{i=1}^n$ do not arrive from a distribution, but instead are chosen by a higher level algorithm, and
(ii) an efficient online scheme is required to track the solution of a least squares regression problem with a growing data set, and thus a drifting target.
Earlier works solve one batch problem or a sequence of batch problems with data arriving from a distribution. On the other hand, we consider a drifting regression setting and study low complexity SGD schemes. For a strongly convex setting, we are able to provide theoretical guarantees, while for a non-strongly convex setting, we obtain encouraging results empirically.

Gradient Descent for Online Least Squares

In this section, we present the results for the fOLS-GD procedure outlined earlier. Recall that fOLS-GD tracks the OLS estimate $\hat{\theta}_n := \min_\theta \frac{1}{2}\sum_{i=1}^n (y_i - \theta^\intercal x_i)^2$ as the samples (x_i, y_i) arrive sequentially (see Fig. 1) and updates the parameter as follows: Fix θ_0 arbitrarily and update

$$\theta_n = \theta_{n-1} + \gamma_n (y_{i_n} - \theta_{n-1}^\intercal x_{i_n}) x_{i_n}, \tag{5}$$

where $i_n \sim \mathcal{U}(\{1, \ldots, n\})$. Here $\mathcal{U}(S)$ denotes the uniform distribution on the set S, and so the samples (x_{i_n}, y_{i_n}) passed to (5) are chosen uniformly randomly from the set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$.

Results We make the following assumptions:
(A1) $\sum_n \gamma_n = \infty$ and $\sum_n \gamma_n^2 < \infty$.
(A2) Boundedness of x_n, i.e., $\sup_n \|x_n\|_2 \leq 1$.
(A3) The noise $\{\xi_n\}$ is i.i.d. and $|\xi_n| \leq 1, \forall n$.
(A4) For all n larger than some initial n_0, $\lambda_{\min}(\bar{A}_n) \geq \mu$, where $\lambda_{\min}(\cdot)$ denotes the smallest eigenvalue of a matrix.
The first assumption is a standard one for the step sizes of SGD, and, more generally, stochastic approximation schemes. While the next two assumptions are standard in the context of least squares, the last assumption is made necessary due to the fact that we do not regularise the problem. Initially A_n may not invertible, and hence the condition can only reasonably hold after some initial time n_0.

In the following, we bound the approximation error $\|\theta_n - \hat{\theta}_n\|$ of fOLS-GD, both in high probability as well as in expectation.

Theorem 1. *Under (A2)-(A4), with $\gamma_n = c/(4(c+n))$ and $\mu c/4 \in (2/3, 1)$, for any $\delta > 0$ and $n > n_0$,*

$$\mathbb{E}\left(\left\|\theta_n - \hat{\theta}_n\right\|_2\right) \leq \frac{K_1(n)}{\sqrt{n+c}}, \text{ and}$$
$$P\left(\left\|\theta_n - \hat{\theta}_n\right\|_2 \leq \frac{K_2(n)}{\sqrt{n+c}}\right) \geq 1 - \delta, \tag{6}$$

where

$$K_1(n) = \frac{\|\theta_{n_0} - \theta^*\| \ln(n_0)}{(n+c)^{\mu c/4}} + \sqrt{h(n)} + \frac{\sqrt{2} + \sqrt{\mu \beta_{n+c}}}{\mu},$$
$$K_2(n) = \sqrt{2K_{\mu,c} \log \frac{1}{\delta}} + K_1(n),$$
$$K_{\mu,c} = c^2 / \left[16\left(1 - 2(1 - 3\mu c/16)\right)\right],$$
$$\beta_n = \max\left(128 d \log n \log n^2 \delta^{-1}, \left(2 \log n^2 \delta^{-1}\right)^2\right),$$
$$h(k) = 2\left[1 + 2(\|\theta_{n_0} - \theta^*\|_2 + \log k)^2\right].$$

Proof Sketch In order to prove the bound in expectation, following the proof scheme of (Frikha and Menozzi 2012), we expand the error at time n into an initial error term, a (martingale) sampling error term, and a drift error term as follows:

$$\begin{aligned}
&\theta_n - \hat{\theta}_n = \theta_n - \hat{\theta}_{n-1} + \hat{\theta}_{n-1} - \hat{\theta}_n \\
=&\theta_{n-1} - \hat{\theta}_{n-1} + \hat{\theta}_{n-1} - \hat{\theta}_n + \gamma_n (y_{i_n} - \theta_{n-1}^\intercal x_{i_n}) x_{i_n} \\
=&\underbrace{\frac{\Pi_n}{\Pi_{n_0}}(\theta_{n_0} - \theta^*)}_{\text{Initial Error}} + \sum_{k=n_0+1}^{n} \underbrace{\gamma_k \frac{\Pi_n}{\Pi_k} \Delta \tilde{M}_k}_{\text{Sampling Error}} - \underbrace{\frac{\Pi_n}{\Pi_k}(\hat{\theta}_k - \hat{\theta}_{k-1})}_{\text{Drift Error}}
\end{aligned}$$

where $\Pi_n := \prod_{k=1}^n \left(I - \gamma_k \bar{A}_k\right)$, and $\Delta \tilde{M}_k$ is a martingale difference (see the Appendix A in (Korda, Prashanth, and Munos 2014) for details).

The initial and sampling errors appear as in previous works on SGD (cf. (Frikha and Menozzi 2012) and (Bach and Moulines 2011)), and can be treated similarly, except that here we can make all the constants explicit, using the specific form of the update rule, and also that $\left\|\Pi_n \Pi_k^{-1}\right\|_2 \leq exp(\Gamma_n - \Gamma_k)$, where $\Gamma_n := \sum_{i=1}^n \gamma_i$. In this way, choosing the step sequence as in the Theorem statement, we derive the first and second terms of $K_1(n)$.

The drift error, however, is not present in previous works, and comes from the fact that the target of the algorithm, $\hat{\theta}_n$, is drifting over time. To control it we note that

$$\left(\nabla F_n(\hat{\theta}_n) = 0 = \nabla F_{n-1}(\hat{\theta}_{n-1})\right) \implies$$
$$\left(\hat{\theta}_{n-1} - \hat{\theta}_n = \left(\xi_n A_{n-1}^{-1} - (x_n^\intercal (\hat{\theta}_n - \theta^*)) A_{n-1}^{-1}\right) x_n\right).$$

Thus it is controlled by the convergence of the least squares solution $\hat{\theta}_n$ to θ^*. Adapting a confidence ball result from (Dani, Hayes, and Kakade 2008), we derive the third term of K_1.

Having bounded the mean error, we can bound separately the deviation of the error from its mean. To do this, following (Frikha and Menozzi 2012), we decompose $\|\theta_n - \hat{\theta}_n\|^2 - E\|\theta_n - \hat{\theta}_n\|^2$ into a sum of martingale differences as follows: Let $\mathcal{H}_n$ denoting the sigma-field $\sigma(i_1, \ldots, i_n)$.

$$\|\theta_n - \hat{\theta}_n\|_2 - \mathbb{E}\|\theta_n - \hat{\theta}_n\|_2 = \sum_{i=1}^n g_i - \mathbb{E}[g_i \,|\mathcal{H}_{i-1}], \tag{7}$$

where $g_i = \mathbb{E}[\|\theta_n - \hat{\theta}_n\|_2 \,|\theta_i]$. Next, we establish that the functions g_i are Lipschitz continuous in the noise ξ_i, with

Lipschitz constants L_i. Unlike in (Frikha and Menozzi 2012) we use the exact form of the update to derive the exact constants L_i. The final step of the proof is to invoke a standard martingale concentration bound. Together with the choice of step size we obtain the first term in K_2. We refer the reader to the Appendix A in (Korda, Prashanth, and Munos 2014) for a detailed proof.

Rates With the step-sizes specified in Theorem 1, we see that the initial error is forgotten exponentially faster than the drift and sampling errors, which vanish at the rate $O\left(n^{-1/2}\right)$. The rate derived in Theorem 1 matches the asymptotically optimal convergence rate for SGD type schemes that do not involve a drifting target (see (Nemirovsky and Yudin 1983)).

Dependence on d The dependence of the rate derived above on the dimension d of x_i is indirect, through the strong convexity constant μ. For example, in the application to strongly-convex linear bandits in the next section, after an initial d steps, the strong convexity constant is known and is of order $\mu = \Omega(1/d)$, and so the derived rate has a linear dependence on d.

Iterate Averaging Ensuring the optimal rate for fOLS-GD requires knowledge of the strong convexity constant μ. In our application to linear bandits in the next section we know this constant. However, we can use Polyak averaging together with the step size $\gamma_n = cn^{-\alpha}$ to arrive at an optimal rate independent of the choice of c.

Strongly Convex Bandits with Online GD

Background for PEGE In this section, we assume that $\mathcal{D}$ is a strongly convex set and the "best action" function, denoted by $G(\theta) := \arg\min_{x\in\mathcal{D}}\{\theta^\intercal x\}$, is assumed to be smooth in the unknown parameter θ that governs the losses of the bandit algorithm (see (A5) below). PEGE of (Rusmevichientong and Tsitsiklis 2010) is a well-known algorithm in this setting. Recall from the introduction that it gathers data and computes least squares estimates of θ^* during exploration phases, between which it exploits the estimates during exploitation phases of growing length. Since strong convexity in the regression problem is guaranteed by the algorithm we propose a variant of PEGE which replaces the calculation of the least squares estimate with fOLS-GD (see Algorithm 1). Whereas, after m exploration phases, PEGE has incurred a complexity of $O(md^3)$, our algorithm has incurred an improved complexity of only $O(md^2)$.

Results We require the following extra assumptions from (Rusmevichientong and Tsitsiklis 2010):
(A4') A basis $\{b_1,\dots,b_d\} \in \mathcal{D}$ for $\mathbb{R}^d$ is known to the algorithm.
(A5) The function $G(\theta)$ is J-Lipschitz.
The assumption (A5) is satisfied, for example, when $\mathcal{D}$ is the unit sphere. However it is not satisfied when $\mathcal{D}$ is discrete. The main result that bounds the regret of fPEGE-GD is given below. The final bound is worse than that for PEGE by only a factor of $O(\log^4(n))$:

Algorithm 1 fPEGE-GD

Input: Get a basis $\{b_1,\dots,b_d\} \in D$ for $\mathbb{R}^d$. Set $c = 4d/(3\lambda_{\min}(\sum_{i=1}^d b_i b_i^\intercal))$ and $\theta_0 = 0$.
for $m = 1, 2, \dots$ **do**
  Exploration Phase
  for $n = (m-1)d$ **to** $md-1$ **do**
    Choose arm $x_n = b_{n \bmod md}$ and observe y_n.
    Update θ as follows: $\theta_n = \theta_{n-1} + \frac{c}{n}((y_j - \theta_{n-1}^\intercal x_j)x_j)$, where $j \sim \mathcal{U}(1,\dots,n)$.
  end for
  Exploitation Phase
  Find $x = G(\theta_{md}) := \arg\min_{x\in D}\{\theta_{md}^\intercal x\}$.
  Choose arm x m times consecutively.
end for

Theorem 2. *Let $\lambda_{PEGE} := \lambda_{\min}(\sum_{i=1}^d b_i b_i^\intercal)$. Under the assumptions (A2), (A3), (A4'), and (A5) and with stepsize $\gamma_n = c/(4(c+n))$, where $c/(4\lambda_{PEGE}) \in (2/3, 1)$, the cumulative regret R_n of fPEGE-GD is bounded as follows:*

$$R_n \le CK_1(n)^2 d^{-1}(\|\theta^*\|_2 + \|\theta^*\|_2^{-1})n^{1/2},$$

where C is a constant depending on λ_{PEGE} and J, and $K_1(n) = O(d\log^2(n))$.

Proof. We have $\lambda_{\min}(\bar{A}_n) \ge \lambda_{\min}\left(\frac{(n \bmod d)\sum_{i=1}^d b_i b_i^\intercal}{[(n \bmod d)+1]d}\right) \ge \lambda_{\min}\left(\sum_{i=1}^d b_i b_i^\intercal\right)/(2d)$ for all $n > d$. So, choosing c as in the theorem statement, we can apply Theorem 1 to get:

$$\mathbb{E}\,\|\theta_n - \theta^*\|_2^2 \le K_1(n)^2/(dn). \tag{8}$$

Now to complete the proof we only need to reprove Lemma 3.6 of (Rusmevichientong and Tsitsiklis 2010), which states that for all $n \ge d$, $\mathbb{E}\,\|\theta^*(G(\theta^*) - G(\theta_{md}))\|_2 \le \frac{K_1(n)}{dm\|\theta^*\|_2}$:

$$\begin{aligned}
&\|\theta^*(G(\theta^*) - G(\theta_{md}))\|_2 \\
=&\big\|(\theta^* - \theta_{md})^\intercal G(\theta^*) + (G(\theta^*) - G(\theta_{md}))^\intercal\theta_{md} \\
&\quad + (\theta_{md} - \theta^*)G(\theta_{md})\big\|_2 \\
\le&\|(\theta^* - \theta_{md})^\intercal(G(\theta^*) - G(\theta_{md}))\|_2 \le \frac{2J\,\|\theta^* - \theta_{md}\|_2^2}{\|\theta^*\|_2},
\end{aligned}$$

where the second inequality we have used that $G(\theta) = G(a\theta)$ for all $a > 0$, (A5), and Lemma 3.5 of (Rusmevichientong and Tsitsiklis 2010).

The rest of the proof follows that of Theorem 3.1 of (Rusmevichientong and Tsitsiklis 2010). □

Online GD for Regularized Least Squares

Ideally an online algorithm would not need to satisfy an assumption such as (A4). Perhaps the most obvious way to obviate (A4) is to regularise. In an offline setting the natural regularisation parameter would be λ/T for some $\lambda > 0$, where T is the size of the batch. However in an online setting we envisage obtaining arbitrary amounts of information,

and so we need to regularize adaptively at each time step by λ_n (see (3)). As outlined earlier, the fRLS-GD algorithm attempts to shadow the solutions $\tilde{\theta}_n$ of the λ_n-regularised problem, using the following iterate update:

$$\theta_n = \theta_{n-1} + \gamma_n((y_{i_n} - \theta_{n-1}^\intercal x_{i_n})x_{i_n} - \lambda_n\theta_{n-1}), \quad (9)$$

where $i_n \sim \mathcal{U}(1,\ldots,n)$.

Discussion It is interesting to note that the analysis in Theorem 1 does not generalise to this setting. Following the same argument as for the proof of Theorem 1 will lead to the iteration:

$$\theta_n - \tilde{\theta}_n = \underbrace{\tilde{\Pi}_n(\theta_{n_0} - \theta^*)}_{\text{Initial Error}} - \underbrace{\sum_{k=1}^{n} \tilde{\Pi}_n\tilde{\Pi}_k^{-1}(\tilde{\theta}_k - \tilde{\theta}_{k-1})}_{\text{Drift Error}} + \underbrace{\sum_{k=1}^{n} \gamma_k\tilde{\Pi}_n\tilde{\Pi}_k^{-1}\Delta\tilde{M}_k}_{\text{Sampling Error}}, \quad (10)$$

where $\tilde{\Pi}_n := \prod_{k=1}^{n}\left(I - \gamma_k(\bar{A}_k + \lambda_k I)\right)$. Under the assumption that we have no control over the smallest eigenvalue of $\bar{A}_k$, we can only upper bound the initial error by $\exp(-\sum_{k=1}^{n}\gamma_k\lambda_k)$. Therefore, in order that the initial error go to zero we must have that $\sum_{k=1}^{n}\gamma_k\lambda_k \to \infty$ as $n \to \infty$. Taking a step size of the form $\gamma_n = O(n^{-\alpha})$ therefore forces $\lambda_n = \Omega(n^{-(1-\alpha)})$. However, examining the drift

$$\begin{aligned}\tilde{\theta}_{n-1} - \tilde{\theta}_n &= \xi_n(A_{n-1} + (n-1)\lambda_{n-1}I)^{-1}x_n \\ &\quad - (x_n^\intercal(\tilde{\theta}_n - \theta^*))(A_{n-1} + (n-1)\lambda_{n-1}I)^{-1}x_n \\ &\quad + ((n-1)\lambda_{n-1} - n\lambda n)(A_{n-1} + (n-1)\lambda_{n-1}I)^{-1}\tilde{\theta}_n.\end{aligned}$$

So when $\lambda_n = \Omega(n^{-(1-\alpha)})$, then we find that $\tilde{\theta}_{n-1} - \tilde{\theta}_n = \Omega(n^{-1})$, whenever $\alpha \in (0,1)$. This, when plugged into (10) results in only a constant bound on the error (note, α must be chosen in $(1/2,1)$ to ensure (A1) holds). Unlike in the setting of (Tarrès and Yao 2011), we do not assume that the data arrive from a distribution, and hence the bias error is difficult to control.

Numerical Experiments

Background for LinUCB In this section the action sets $\mathcal{D}_n \subset \mathcal{D}$ are finite, but possibly varying. A popular algorithm for such settings is the LinUCB algorithm. This algorithm calculates UCBs for the mean reward obtained by choosing each individual feature in $\mathcal{D}_n$ as follows:

$$\forall x \in \mathcal{D}_n, \quad UCB(x) := x^\intercal\hat{\theta}_n + \kappa\sqrt{x^\intercal A_n^{-1}x},$$

where κ is a parameter set by the agent that can be understood to be controlling the rate of exploration the algorithm performs. Having calculated the UCBs for all available features the agent then chooses the feature with the highest UCB. LinUCB needs to compute online the inverse of the matrix A_n^{-1} in order to compute the UCBs for each iteration of the algorithm, and so we propose improving the complexity by using an SGD scheme to approximate the UCBs. Since LinUCB cannot guarantee strong convexity of the regression problem, we investigate experimentally applying the regularised fRLS-GD in place of RLS solutions.

Tracking the UCBs While we can track the regularised estimates $\tilde{\theta}_n$ using fRLS-GD as given above, to track the UCBs we derive the analogous update rule for each feature $x \in \mathcal{D}_n$:

$$\phi_n = \phi_{n-1} + \gamma_n((n^{-1}x - ((\phi_{n-1})^\intercal x_{(i_n)})x_{(i_n)})), \quad (11)$$

where $i_n \sim \mathcal{U}(1,\ldots,n)$. The UCB value corresponding to feature x is then set as follows:

$$UCB(x) := x^\intercal\theta_n + \kappa\sqrt{x^\intercal\phi_n}.$$

If the action sets $\mathcal{D}_n$ were fixed (say $\mathcal{D}_1$), then we take one step according to (11) for each arm $x \in \mathcal{D}_1$ in each iteration n of LinUCB. The computational cost of this LinUCB variant is of order $O(|\mathcal{D}_1|dn)$, as opposed to the $O(d^2n)$ incurred by the vanilla LinUCB algorithm that directly calculates A_n^{-1}. This variant would give good computational gains when $|\mathcal{D}_1| \ll d$. If the action sets change with time, then one can perform a batch update, i.e., run T steps according to (11) for each feature $x \in \mathcal{D}_n$ in iteration n of LinUCB. This would incur a computational complexity of order $O(KTdn)$, where K is an upper bound on $|\mathcal{D}_n|$ for all n, and result in good computational gains when $KT \ll d$.

Simulation Setup. We perform experiments on a news article recommendation platform provided for the ICML exploration and exploitation challenge ((Mary et al. 2012)). This platform is based on the user click log dataset from the Yahoo! front page, provided under the Webscope program ((Webscope 2011)). An algorithm for this platform is required to repeatedly select a news article from a pool of articles and show them to users. Each article-user pair is described in the dataset by a feature vector, which the algorithm can use to make its decisions. We implement the LinUCB algorithm (popular for this setting) as well as three SGD variants as described below.

(i) fLinUCB-GD: This is described in Algorithm 2 and uses fRLS-GD in place of RLS.

(ii) fLinUCB-SVRG: This is similar to the above algorithm, except that the SGD scheme used is derived from (Johnson and Zhang 2013).

(iii) fLinUCB-SAG: This is a variant that uses the SGD scheme proposed by (Roux, Schmidt, and Bach 2012).

The last two algorithms above are based on two recent approaches for accelerating the convergence of SGD-type schemes (see Appendices B and C of (Korda, Prashanth, and Munos 2014) for a description).

Results We use tracking error and runtimes as performance metrics for comparing the algorithms. The tracking error is the difference in ℓ^2 norm between the SGD iterate θ_n and RLS solution $\tilde{\theta}_n$, at each instant n of the SGD variant of LinUCB.

Algorithm 2 fLinUCB-GD

Initialisation: Set θ_0, γ_k - the step-size sequence.
for $n = 1, 2, \ldots$ **do**
  Approximate RLS
    Observe article features $x_n^{(1)}, \ldots, x_n^{(K)}$
    Approximate $\hat{\theta}_n$ using fRLS-GD iterate θ_n (4)
  UCB computation
    for $k = 1, \ldots, K$ **do**
      Estimate confidence parameter $\phi_n^{(k)}$ using (11)
      Set $\text{UCB}(x_n^{(k)}) := \theta_n^\intercal x_n^{(k)} + \kappa\sqrt{{x_n^{(k)}}^\intercal \phi_n^{(k)}}$
    end for
  Choose article $\arg\max_{k=1,\ldots,K} \text{UCB}(x_n^{(k)})$ and observe the reward.
end for

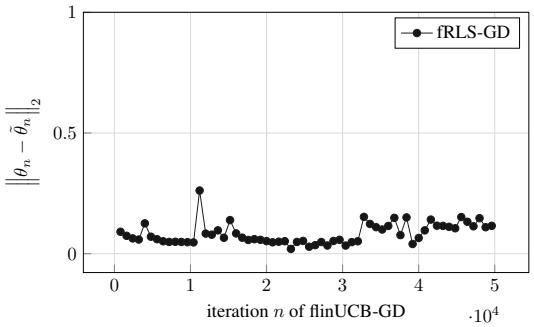
Figs. 2(a)–2(c) present the tracking error with day 2's data file as input for fRLS-GD, SVRG and SAG variants of LinUCB, respectively. It is evident that all the SGD schemes track the corresponding RLS solutions consistently. Fig. 2(d) report the runtimes observed on two different data files corresponding to days 2 and 4 in October, 2009 (see (Webscope 2011)) of the dataset. It is evident that the SGD schemes result in significant computational gains in comparison to classic RLS solvers (e.g. Sherman-Morrison lemma).

Finally, we observed that the SGD variants under best configurations achieved 75% of the regular LinUCB CTR score. CTR score is the ratio of the number of clicks an algorithm gets to the total number of iterations it completes, multiplied by 10000. Considering that the dataset contains very sparse features and also the fact that the rewards are binary, with a reward of 1 occurring rarely, we believe LinUCB has not seen enough data to have converged UCB values and hence the observed loss in CTR may not be conclusive.

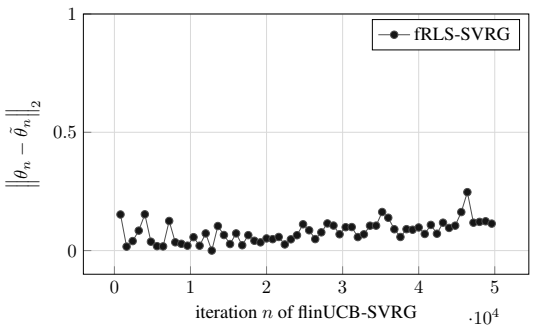
Conclusions

We analysed online SGD schemes for the problem of drifting least squares regression problems in the context of a higher level algorithm. In particular, when the higher level algorithm can guarantee strong convexity in the data, we provided error bounds both in expectation and in high probability. Further, we derived an SGD variant of PEGE linear bandit algorithm with a speed up of $O(d)$ at the cost of only logarithmic factors in the regret. For the non-strongly convex setting, we studied an adaptively regularised SGD scheme by combining it with LinUCB. The empirical results of this algorithm on a large-scale news recommendation application are encouraging. However a theoretical analysis of the adaptively regularised SGD scheme remains challenging, and is an interesting direction for future work.

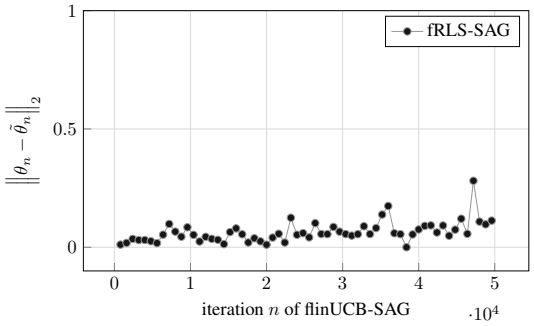
Acknowledgments The first author was gratefully supported by the EPSRC project, Autonomous Intelligent Systems EP/I011587. The second and third authors would like to thank the European Community's Seventh Framework Programme (FP7/2007 − 2013) under grant agreement n^o 270327 for funding the research leading to these results.



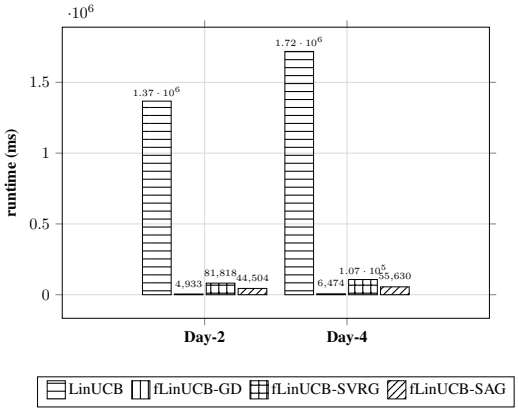
(a) Tracking error: fRLS-GD



(b) Tracking error: fRLS-SVRG



(c) Tracking error: fRLS-SAG



(d) Runtimes (in ms) on two days of the dataset

Figure 2: Performance evaluation of fast LinUCB variants

References

Bach, F., and Moulines, E. 2011. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. *Advances in Neural Information Processing Systems (NIPS)*.

Dani, V.; Hayes, T. P.; and Kakade, S. M. 2008. Stochastic linear optimization under bandit feedback. In *Proceedings of the 21st Annual Conference on Learning Theory (COLT)*, 355–366.

Frikha, N., and Menozzi, S. 2012. Concentration Bounds for Stochastic Approximations. *Electron. Commun. Probab.* 17:1–15.

Hazan, E., and Kale, S. 2011. Beyond the regret minimization barrier: an optimal algorithm for stochastic strongly-convex optimization. *Journal of Machine Learning Research* 19:421–436.

Johnson, R., and Zhang, T. 2013. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems (NIPS)*, 315–323.

Korda, N.; Prashanth, L.; and Munos, R. 2014. Fast gradient descent for least squares regression: Non-asymptotic bounds and application to bandits. *arXiv preprint arXiv:1307.3176v4*.

Li, L.; Chu, W.; Langford, J.; and Schapire, R. E. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, 661–670. ACM.

Mary, J.; Garivier, A.; Li, L.; Munos, R.; Nicol, O.; Ortner, R.; and Preux, P. 2012. ICML Exploration and Exploitation 3 - New Challenges.

Nemirovsky, A., and Yudin, D. 1983. Problem complexity and method efficiency in optimization.

Rakhlin, A.; Shamir, O.; and Sridharan, K. 2011. Making gradient descent optimal for strongly convex stochastic optimization. *arXiv preprint arXiv:1109.5647*.

Roux, N. L.; Schmidt, M.; and Bach, F. 2012. A stochastic gradient method with an exponential convergence rate for finite training sets. *arXiv preprint arXiv:1202.6258*.

Rusmevichientong, P., and Tsitsiklis, J. N. 2010. Linearly parameterized bandits. *Math. Oper. Res.* 35(2):395–411.

Shalev-Shwartz, S., and Zhang, T. 2012. Stochastic dual coordinate ascent methods for regularized loss minimization. *arXiv preprint arXiv:1209.1873*.

Tarrès, P., and Yao, Y. 2011. Online learning as stochastic approximation of regularization paths. *arXiv preprint arXiv:1103.5538*.

Webscope, Y. 2011. Yahoo! webscope dataset ydata-frontpage-todaymodule-clicks-v2_0.

Zinkevich, M. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th International Conference on Machine Learning*, 928–925.